CodeSpider: Automatic Code Querying with Multi-modal Conjunctive Query Synthesis

Chengpeng Wang      Prism Group, HKUST
Code Querying

• Development assistance
  • How a specific class is used?

• Patch generation
  • Where is log4j interface invoked?

• Code measurement
  • How many projects import log4j as an external library?
Querying Code in IDEs

- **IDEs**
  - Eclipse: String matching
  - IntelliJ: Structural searching

- Restrictive Search Template
Querying Code with Datalog

- Datalog-based program analyzer, e.g., CodeQL
  - Write a Datalog-like query program to specify the querying condition

- Example
  - Find all the assignments from float to integer variables

```datalog
from AssignExpr a
where a.getRValue().getType() instanceof FloatingPointType
  and a.getLValue().getType() instanceof IntegralType
select a
```

+ Advanced Querying Support
  - Heavy Learning Burden
  -Verbose Query Writing
Our Aim: A Better Way

- Automatic synthesizing a conjunctive query

### Methods receiving a parameter with Log4jUtils type.

<table>
<thead>
<tr>
<th>Positive Example</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>public void foo(Log4jUtils a)</td>
<td>{ return; }</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Negative Example</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>private void goo(int a)</td>
<td>{ return; }</td>
</tr>
</tbody>
</table>

### Query

```datalog
query(Method m) :-
    exists(Parameter p, Type t, String s)
    p = m.getPara() &&
    t = p.getType() &&
    s = t.getName() &&
    equals(s, "Log4jUtils")
```

**Ease of use:** Use Datalog-based analyzers as a black box  
**Capability:** Leverage various relations describing program properties
Preliminary: Relational Representation

// positive example
public void foo(Log4jUtils a) {
    return;
}
// negative example
private void goo(int a) {
    return;
}

Semantic constraint:
Separating positive tuples from negative tuples
Challenges

• Incredibly large search space
  • Large numbers of relations
  • Flexible combination of relations

• Multiple candidates satisfying the semantic constraint
  • Ineffective selection introduces the over-fitting problem
Stage I: Sketch Generation

• Summarize query sketches by the subgraphs of TTN
  • TTN encodes the type information of the attributes in each relation

Type Transition Net (TTN)

query(Method m) :-
  exists(Parameter p, Type t)
  m = p.getMethod() &&
  t = p.getType()
Stage II: Query Refinement

- Obtain all the query candidates after query refinement
  - Add atomic formulas until positive and negative tuples are separated.
  - Discard the query if it misses a positive example.

`query(Method m) :- true`

`query(Method m) :- exists(Parameter p, Type t)
  m = p.getMethod() &&
  t = p.getType() &&
  s = t.getName() &&
  equals(s, "Log4jUtils")`

`query(Method m) :-
  exists(Parameter p, Type t, String s)
  m = p.getMethod() &&
  t = p.getType() &&
  s = t.getName() &&
  equals(s, "Log4jUtils")`
Stage III: Query Selection

- Select the query covering the entities in the NL description as many as possible with a simple form
  - Dual metrics: Entity coverage ($\alpha$), Structural complexity ($\beta$)

\[
\begin{align*}
\text{NL Description: Methods receiving a parameter with Log4jUtils type.} \\
\text{query(Method m) :-} & \exists \text{Parameter p, Type t, String s} \\
& p = m.getPara() \land \land \\
& t = p.getType() \land \land \\
& s = t.getName() \land \land \\
& \text{equals}(s, "Log4jUtils") \\
& \alpha = 1/3 \\
& \beta = 2 \\
& \text{query(Method m) :-} & \exists \text{Parameter p, Type t, Modifier f, String s1, String s2} \\
& p = m.getPara() \land \land t = p.getType() \land \land \\
& s1 = t.getName() \land \land \text{equals}(s1, "Log4jUtils") \land \land \\
& f = m.getModifier() \land \land s2 = f.getName() \land \land \\
& \text{equals}(s2, "public") \\
& \alpha = 3/3 \\
& \beta = 4 \\
& \text{query(Method m) :-} & \exists \text{String s} \\
& s = m.getName() \land \land \\
& \text{equals}(s, "foo") \\
& \alpha = 3/3 \\
& \beta = 7
\end{align*}
\]
Another Perspective

Find the best abstraction for given tuples:

- Syntax: Conjunctive query
- Soundness: Cover positive tuples and exclude negative ones
- Optimality: Optimize the dual metrics

Examples
Syntax
NL Description

Tuples
Template
Dual metrics

Examples
Syntax
NL Description

Tuples
Template
Dual metrics

- positive tuple
- negative tuple
Implementation

• Implement *CodeSpider* in Python
  • Leverage GSA (General Suffix Automaton) to guide the synthesis of string constraints
    • Support the string predicates, including `prefixOf`, `suffixOf`, `equals`, and `contains`.
  • CodeSpider supports synthesizing queries for Sparrow, a commercial Datalog-based analyzer developed by Ant Group.
    • 173 relations with 1,093 attributes
Evaluation: Capability

• Code querying tasks

<table>
<thead>
<tr>
<th>ID</th>
<th>Description</th>
<th>#P, #N</th>
<th>#C, #A</th>
<th>Kind</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Float variables of which the identifier contains “cash”</td>
<td>(3, 1)</td>
<td>(4, 4)</td>
<td>Var</td>
</tr>
<tr>
<td>2</td>
<td>Cast expressions from double-type to float type</td>
<td>(1, 2)</td>
<td>(6, 7)</td>
<td>Expr</td>
</tr>
<tr>
<td>3</td>
<td>Expressions comparing long int with int</td>
<td>(1, 2)</td>
<td>(3, 6)</td>
<td>Expr</td>
</tr>
<tr>
<td>4</td>
<td>Cast expressions casting long to int</td>
<td>(2, 1)</td>
<td>(6, 7)</td>
<td>Expr</td>
</tr>
<tr>
<td>5</td>
<td>Expressions comparing a variable and Boolean literal</td>
<td>(1, 3)</td>
<td>(4, 5)</td>
<td>Expr</td>
</tr>
<tr>
<td>6</td>
<td>New expressions of ArrayList</td>
<td>(1, 1)</td>
<td>(3, 3)</td>
<td>Expr</td>
</tr>
<tr>
<td>7</td>
<td>Logical-and expressions with literal as an operand</td>
<td>(2, 2)</td>
<td>(4, 5)</td>
<td>Expr</td>
</tr>
<tr>
<td>8</td>
<td>The import of LocalTime</td>
<td>(2, 1)</td>
<td>(3, 4)</td>
<td>Stmt</td>
</tr>
<tr>
<td>9</td>
<td>The import of the classes in log4j</td>
<td>(1, 1)</td>
<td>(2, 2)</td>
<td>Stmt</td>
</tr>
<tr>
<td>10</td>
<td>Labeled statements</td>
<td>(2, 2)</td>
<td>(1, 0)</td>
<td>Stmt</td>
</tr>
<tr>
<td>11</td>
<td>If-statements with a Boolean literal as a condition</td>
<td>(2, 1)</td>
<td>(2, 1)</td>
<td>Stmt</td>
</tr>
<tr>
<td>12</td>
<td>For-statements with a Boolean literal as a condition</td>
<td>(2, 1)</td>
<td>(2, 1)</td>
<td>Stmt</td>
</tr>
<tr>
<td>13</td>
<td>Public methods with void return type</td>
<td>(2, 1)</td>
<td>(5, 6)</td>
<td>Method</td>
</tr>
<tr>
<td>14</td>
<td>Methods receiving a parameter with Log4jUtils type</td>
<td>(2, 1)</td>
<td>(4, 4)</td>
<td>Method</td>
</tr>
<tr>
<td>15</td>
<td>Classes with a login method</td>
<td>(2, 1)</td>
<td>(3, 3)</td>
<td>Class</td>
</tr>
<tr>
<td>16</td>
<td>Classes containing a field with float type</td>
<td>(1, 1)</td>
<td>(4, 4)</td>
<td>Class</td>
</tr>
</tbody>
</table>
Evaluation: High Efficiency

- Average time cost: 3.35 seconds
- Maximal time cost: 8.91 seconds
- Minimal time cost: 2.23 seconds
- 14 tasks finished in 4 seconds
Conclusion

• (Conceptual) We define a multi-modal program synthesis problem for code querying.
• (Technical) We propose an efficient algorithm for synthesizing a conjunctive query.
• (Empirical) We evaluate our synthesis algorithm upon real-world code querying tasks and obtain the target queries efficiently.
Thank you for your listening!